OOI – Cyberinfrastructure

Data Management Subsystem Pilot Period Report

Version 1-00
Draft

Consortium for Ocean Leadership
1201 New York Ave NW, 4th Floor, Washington DC 20005
www.OceanLeadership.org

in Cooperation with

University of California, San Diego
University of Washington
Woods Hole Oceanographic Institution
Oregon State University
Scripps Institution of Oceanography
Document Control Sheet

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
<th>Originator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-01</td>
<td>Dec 11, 2009</td>
<td>Initial</td>
<td>M. Meisinger</td>
</tr>
<tr>
<td>0-02</td>
<td>Dec 17, 2009</td>
<td>Content by P. Hubbard, L. Bermudez</td>
<td>D. Stuebe</td>
</tr>
<tr>
<td>0-03</td>
<td>Dec 17, 2009</td>
<td>Revision</td>
<td>D. Stuebe</td>
</tr>
<tr>
<td>0-04</td>
<td>Dec 18, 2009</td>
<td>Revision</td>
<td>M. Meisinger</td>
</tr>
<tr>
<td>0-05</td>
<td>Dec 28, 2009</td>
<td>Revision</td>
<td>J. Gallagher</td>
</tr>
<tr>
<td>0-06</td>
<td>Dec 31, 2009</td>
<td>Revision</td>
<td>L. Bermudez and J. Graybeal</td>
</tr>
<tr>
<td>0-07</td>
<td>Jan 04, 2010</td>
<td>Revision</td>
<td>David Stuebe</td>
</tr>
<tr>
<td>0-08</td>
<td>Jan 06, 2010</td>
<td>Formatting revision</td>
<td>M. Meisinger</td>
</tr>
</tbody>
</table>

Document Information

<table>
<thead>
<tr>
<th>Project</th>
<th>Ocean Observatories Infrastructure (OOI) CyberInfrastructure (CI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Custodian</td>
<td>OOI CI Architecture &amp; Design Team (ADT) Michael Meisinger (UCSD), OOI CI Senior System Architect</td>
</tr>
<tr>
<td>Approval</td>
<td>Alan Chave (WHOI), OOI CI System Engineer Matthew Arrott (UCSD), OOI CI Project Manager</td>
</tr>
<tr>
<td>Created on</td>
<td>December 11, 2009</td>
</tr>
<tr>
<td>Last Changed on</td>
<td></td>
</tr>
<tr>
<td>Document Status</td>
<td>Candidate Draft</td>
</tr>
</tbody>
</table>
Table of Contents

1 INTRODUCTION ........................................................................................................................................... 1

2 DATA MANAGEMENT SUBSYSTEM ........................................................................................................... 1

3 DATA EXCHANGE PROTOTYPE .................................................................................................................. 2
   3.1 TARGETED TECHNOLOGIES .................................................................................................................. 2
   3.2 STEP 1: ERDDAP ON AMAZON EC2 .................................................................................................. 4
   3.3 STEP 2: DATA EXCHANGE ON EC2 ................................................................................................. 6
   3.4 DATA EXCHANGE IMPLEMENTATION ............................................................................................... 8
   3.5 RESULTS AND LESSONS LEARNED .................................................................................................. 10

4 ATTRIBUTE STORE .................................................................................................................................... 11

5 SEMANTIC FRAMEWORK INTEGRATION PROTOTYPE ............................................................................. 13
   5.1 INTRODUCTION TO THE SEMANTIC FRAMEWORK ....................................................................... 14
   5.1.1 Semantic heterogeneity and categorizations—scenario ................................................................. 14
   5.1.2 Risk Reduction .................................................................................................................................. 14
   5.1.3 Summary Semantic Framework .................................................................................................... 16
   5.2 COMPONENTS ...................................................................................................................................... 17
   5.2.1 MMI registry ...................................................................................................................................... 17
   5.2.2 Harvester .......................................................................................................................................... 18
   5.2.3 Faceted Browser ............................................................................................................................... 18
   5.2.4 Metadata Registry ............................................................................................................................. 19
   5.2.5 Community Collaborative Ontology Editor (example: Semantic Media Wiki) .......................... 19
   5.3 TECHNOLOGY EVALUATION – CORE ONTOLOGY .......................................................................... 21
   5.3.1 Technology Evaluation - Ontology Editors ..................................................................................... 21
   5.3.2 Collaborative Protege ...................................................................................................................... 21
   5.3.3 With respect to our review criteria: ................................................................................................. 22
   5.3.4 Semantic MediaWiki ....................................................................................................................... 22
   5.3.5 With respect to our review criteria: ................................................................................................. 22
   5.3.6 Knoodle ........................................................................................................................................... 22
   5.3.7 With respect to our review criteria: ................................................................................................. 23
   5.4 FUTURE DEVELOPMENT STRATEGIES ............................................................................................. 23

6 HYRAK AND GRIDFIELDS INTEGRATION PROTOTYPE ............................................................................ 24
   6.1 GRID FIELDS AS A BACKEND SERVICE ............................................................................................ 25
   6.1.1 Why Unstructured Grids are Complicated ..................................................................................... 25
   6.1.2 Mapping the DAP data model to GridFields Data model ............................................................... 26
   6.2 DESIGN OF THE HYRAK - GRIDFIELDS SERVICE .......................................................................... 26
   6.3 SEPARATING PROTOCOL FROM TRANSPORT IN AMQP ............................................................... 27
   6.3.1 How the OLFS connects to the BES ................................................................................................. 28
   6.3.2 Abstracting the OLFS/BES connection logic ............................................................................... 29
   6.3.3 Abstracting the request-response logic of the OLFS .................................................................... 29
   6.3.4 How complex would this be to implement? .................................................................................. 31

7 SUMMARY .................................................................................................................................................. 31

8 REFERENCES ................................................................................................................................................. 33
1 Introduction

The Data Management (DM) subsystem of the OOI Cyberinfrastructure provides the middle layer of services that link observatory oriented applications with infrastructure services. Data are the linking element between sensors, analyses, numerical models and interactive ocean observing. Managing data and any kind of information is a critical and central component in an observatory.

The DM subsystem combines proven technologies from academic and commercial environments with the development of new abstract interfaces and services which are the binding layer required to integrate the selected technologies. The DM provides:

- Manage and present data and metadata supporting the OOI domain and data models. This covers both data distribution and data storage.
- Policy-governed data access
- User-defined data presentation
- Provision, manage and present data repositories, collections and streams. Support federation and delegation.
- Maintain and ensure the integrity of data in perpetuity
- Complex querying across and integration of geospatial, temporal, spatiotemporal, relational, XML, and ontological (tree and graph structures) resources (mediation)
- Present, find, exploit and annotate data based on a semantic frame of reference
- Provision and exploit sharable semantic frames of reference
- Provision and exploit sharable mappings between different semantic frames of reference (i.e. crosswalks between multiple ontologies)

The highest risks associated with the DM subsystem are due to the pervasive nature of the services, interfaces and data formats which it provides. These issues are addressed in the OOI pilot period that followed the OOI Final Design Review in November 2008 and will be ongoing until the end of December 2009. The pilot period’s goals are to prepare for OOI construction and to mitigate significant risks through prototyping.

This report documents the risk mitigation efforts since January 2009 and their results. The culmination of this effort is a refinement of the DM FDR baseline architecture. The pilot efforts include the Data Exchange (DX) prototype with the Attribute Store, the Semantic Framework Integration Prototype and the Hyrax/GridFields integration prototype.

2 Data Management Subsystem

The services provided by the Data Management subsystem (see Fig. 1) support a wide range of scientific data and information management services; both supporting science data-oriented applications and providing core infrastructure. The ingestion service handles incoming observational and external data ingestion into CI managed data repositories along with any associated metadata. The Transformation service provides syntactical data format transformations as well as ontology-based data mediation, providing semantically enabled access to any data in repositories based on metadata as part of user provided search criteria.
From the point of view of science applications, the critical services are ingestion and transformation. The Ingestion service is responsible for initial data parsing, initial metadata extraction, registration, and versioning of data products received. The Transformation service manages the data content format conversion/ transformation, mediation between syntax and semantics of data (based on ontologies), basic data calibration and QA/QC, additional metadata extraction, qualification, verification and validation. The Presentation service enables data discovery, access, reporting, and branding of data products. For data discovery, it provides the mechanisms to both browse/navigate to specific data products and search/query for them based on specific metadata or data content.

At the infrastructure level, there are services that provide data distribution, preservation and inventory capabilities. The Data Distribution Network—the Exchange—represents the main Data Management infrastructure element that enables the science data distribution throughout the Integrated Observatory network. It represents the integrating data distribution infrastructure that most of the Data Management services are relying on; it itself relies significantly on services from the Common Operating Infrastructure, in particular the Messaging Service, as described in the next section and in [5].

The infrastructure services provide information distribution, persistence and access across space and time, making data obtainable across the observatory network. Information includes observational data and derived data products along with other information resources required for the operation of the integrated observatory, such as ancillary instrument information, user identities, workflow definitions, and executable virtual machine images. The Preservation service is responsible for data replication, preservation, and archival/backup as defined by OOI policies. It also provides distributed data repository capabilities based on the underlying services of the COI subsystem. The Inventory service provides the cataloging, indexing and metadata management capabilities required for data ingestion and retrieval.

3 Data Exchange Prototype

The Data Exchange prototype is a collaboration between the OOI CI and NOAA IOOS in the context of the DIF (Data Integration Framework) project.

3.1 Targeted Technologies

The following technologies are relevant for the purposes of the Data Exchange prototypes described in this report. As part of risk mitigation activities, we have investigated and integrated the following technologies:

- **OPeNDAP Data Access Protocol (DAP)** is a protocol/service for accessing data from large scientific datasets over the Internet (see [11], [22]). Datasets can be queried to return the data types, sizes and at-
tributes of the variables they contain, and then just selected variables and subsets of data can be returned. It was originally designed to work with large amounts of remote sensing data efficiently over the internet, and has been used heavily in the meteorology and oceanographic communities for the last 15 years. The DAP service has been implemented in numerous data delivery systems (e.g. GRADS Data Server, Hyrax Server, PyDAP, THREDDS Data Server), in leading scientific analysis environments (Python, Matlab, IDL, FERRET, GRADS) as well as other applications (e.g. Unidata Integrated Data Viewer, Panoply, ncWMS, ERDDAP).

ERDDAP [16] was built by NOAA ERD to solve some very practical problems: How to translate data between a variety of scientific data formats and services (e.g. NetCDF files, DAP services, Sensor Observation Service). ERDDAP is a RESTful service that provides a simple, consistent way to access data from a variety of data transport services (e.g. DAP, DiGIR, SOS) and return the data in formats that are used in real applications (e.g. .mat, .csv, .kml, .nc, .png), including web development formats (e.g. json, geojson, .htmlTable) ERDDAP also acts as a DAP, SOS and Web Mapping Service (WMS) server, so that a DAP, SOS or WMS client can access data using these protocols from any of the data sources ERDDAP knows about, translating the protocol of the data source as needed. Thus, using ERDDAP, users can, for example, request a subset of gridded data from a DAP server and deliver a NetCDF file or request time series data via SOS and deliver an Excel Spreadsheet. Along with the requested data or plot, users are supplied the URL that accomplishes the requested task, so that they can easily create scripting applications or powerful processing chains. Gridded data must have uniformly spaced geographic coordinates to work with ERDDAP.

THREDDS Data Server (TDS) by Unidata [23] is a Tomcat application that allows a variety of data from a variety of data formats and services to be served via DAP, the OGC Web Coverage Service and the OGC Web Map Service. It supports serving HDF4, HDF5, GRIB, GRIB2, NetCDF3 and NetCDF4 files, as well as virtual datasets constructed with the NetCDF Markup Language (NcML). NcML allows non-standard datasets to be standardized through the addition or correction of metadata as well as a variety of aggregation capabilities. The Ferret THREDDS Data Server (F-TDS) [17] is a customized version of Unidata’s THREDDS Data Server; it provides server-side functions such as time averaging and regridding to be specified in the URL. Using the F-TDS, for instance, a user who accesses a dataset of daily temperature data who wants only the annual mean can specify this calculation on the URL, saving tremendous bandwidth but placing additional computation demands on the server.

GridFields [14] is a library for the algebraic manipulation of scientific datasets. This includes in particular unstructured gridded datasets. Unstructured grids use a mesh of polygons to describe complex structures. One of the benefits of using polygons is the flexibility of resolution and topology. The irregular contours of a river bed or coastline naturally lend themselves to unstructured gridded data representations [Fig 2]. In structured grids, much of the resolution or space would be lost to areas of little interest. One of the downsides of unstructured gridded data is the difficulty of working with it. GridFields provides the library to do so more efficiently.
Amazon Web Services – starting in early 2006, Amazon provided a set of computational and storage capabilities for third party applications [4]. The mechanisms for provisioning and controlling these resources rely on standard Web Services technologies. The major advantage compared with traditional solutions is that cost depends on the actual usage instead of high upfront costs for hardware that might not be used all time. Additional benefits come from the reliability and redundancy of the provided solution and the on-demand scaling with user/application demands. It also allows for rapid prototyping of new technologies with full flexibility in choosing the right architecture for the target application domain.

### 3.2 Step 1: ERDDAP on Amazon EC2

A first prototype was developed for FDR and released with refinements in January 2009 providing web-based user access to oceanographic data through NOAA’s ERDDAP, as well as efficient science metadata caching, and scalable deployment in a cloud computing environment using Amazon’s EC2. Fig. 3 depicts the architecture of this prototype [19].

![Fig. 3. ERDDAP cloud prototype architecture](image)
The central functions of the prototype are provided by ERDDAP (depicted as ERDDAP Utility) as described above. ERDDAP is a server-hosted Java application that provides a web interface for registering data sources (datasets) that are available via HTTP through the Internet. It also provides a web interface to list all available data sources, show their detailed metadata and to query for specific datasets and specific variables. Data consumers can also filter the data sources to the subsets of interest and retrieve the available data sources in a number of formats that ERDDAP can provide. This includes DAP datasets and visualization graphs.

ERDDAP is designed as a single server application. There exists no scaling strategy for scenarios where the user requested transformations and the resulting server load exceeds the capabilities of the server. Although it is possible to add multiple ERDDAP servers behind a load-balancer, this option has the disadvantage of causing higher load on the data sources that are repeatedly queried for updates and may lead to potential inconsistent state between all ERDDAP instances at one given instant, leading to a confusing experience for the users.

Even if multiple instances can be provided, further scalability problems arise eventually through the limitation of computational resources at the site where the ERDDAP application is running. Typically, the hardware environment, such as a grid cluster, is designed with a specific load estimate and has a predetermined capacity. Additional computations cannot be sustained.

The OOI strategy is to use cloud resources to deploy all components of the architecture as a service. This strategy enables the allocation of computational 'instances' on the cloud on demand. For instance, if we detect that ERDDAP utility servers are under high load, we can add another instance to the existing cluster with little or no disruption to the existing system. We designed the prototype to allow adding and removing instances of each component flexibly when needed.

In addition, we designed the components such that their load characteristics are optimally reflected by our scaling strategy. For instance, we split the ERDDAP application into three different types of processes: (1) The ERDDAP utility that provides the web interfaces and the transformation engine, (2) the ERDDAP crawler, which regularly queries the data sources for updates of data and metadata, and (3) the Memcache component that provides the distributed shared state between all instances efficiently. The Metadata and user storage component was realized as a MySQL cluster, allowing for multiple instances of MySQL engine and storage processes. Other types of processes, such as a scheduler, the software load-balancer, and the message broker completed the architecture.

Using the cloud also relieves us of common operations concerns such as power and cooling management, hardware failure, personnel support and network setup. Fig. 8 shows the deployment view on the prototype, with components in our local hardware environment, with virtual machine images within Amazon’s S3 storage cloud, and running instances connected by the message-broker infrastructure on the EC2 cloud.

A local provisioner process is responsible for starting and controlling the entire system. Startup of a typical configuration consisting of 15 different virtual machine images takes less than 2 minutes as part of a fully automated process. A single web URL provided the users’ entry point to the system. Fig. 4 illustrates this process.
Fig. 4. ERDDAP cloud prototype deployment scenario

Fig. 5 shows a detailed view of the components at the operator's location and in the Amazon cloud that enable the automatic provisioning process.

Fig. 5. Prototype EC2 cloud provisioning deployment strategy

3.3 Step 2: Data Exchange on EC2
The primary objective of the second prototype is to deploy a scalable “Data Exchange” infrastructure, leveraging the findings and technologies of the first prototype described in the previous section [20]. The Data Exchange prototype, currently in development, will provide a server-side data processing capability for use by an initial set of active ocean modeling communities to efficiently exchange large model datasets in whole or in part while preserving the original content and structure of the dataset. The targeted
modeling communities participating in this effort include NERACOOS, MARCOOS and SCCOOS. In a subsequent phase the Data Exchange will be promoted for broader use by the IOOS community.

Arising from this primary objective, the effort will provide the IOOS DIF with a platform on which to test the Web Services and Data Encoding being developed to distribute the seven “Core Variables” (Currents, Water Level, Sea Temperature, Salinity/Conductivity, Surface Winds, Waves and Chlorophyll) to their four initial target groups. The process of developing and deploying the Data Exchange in the context of operational user communities will drive the refinement of the OOI Cyberinfrastructure requirements, design and technology choices prior to the start of construction of the OOI. Finally this effort will also provide the IOOS DMAC with practical insight into viable strategies for realizing an integrated national ocean observing cyberinfrastructure.

The Data Exchange’s central premise is to provide modeling communities with an effective community infrastructure to publish datasets and server-side functions that:

1. Register and transmit Datasets of any structural type supported by DAP,
2. Register and transmit Virtual Datasets authored using NcML,
3. Register and execute “Ferret” conformant server-side Functions,
4. Register and trigger Subscriptions that follow the evolution of a Dataset,
5. Register and execute “Data Exchange” conformant Tasks (such as server-side analyses),
6. Link data subscription notifications to the execution of a “Data Exchange” Task,
7. Register and manage “Data Exchange” Communities to delineate and control access to Datasets, Functions, Subscriptions and Tasks.

The Data Exchange as community infrastructure will focus on the following core concerns:

1. Transparency - Existing publishers and consumers will be able to use the infrastructure without making changes to their current practices and processes,
2. Elasticity - The infrastructure will automatically adjust its computing and storage capacity to meet demand
3. Fault-Tolerance – The infrastructure will continue to operate and self-heal in the presence of any infrastructure component failure; i.e., network, storage, computer and/or process.

At the logical level, the Data Exchange prototype provides a collection of services for Data Retrieval, Operations and Management, Front-end capabilities, Data and Metadata Caching with Storage and Retrieval Capabilities (see Fig. 10). There could be multiple instances of a given service at any moment in time. For instance, consider multiple Fetchers implementing transparent access mechanisms to external data sources and scaling with the demand. The Data Exchange relies on a number of infrastructure capabilities such as the Messaging Service, the Storage Repository, and the Attribute Store, which are abstract services provided as standalone components for reuse purposes. Fault tolerance is provided both through replication of necessary capabilities under the control of the Resource Manager and Scheduler, and underlying reliable messaging capabilities of the Messaging Service.

The Data Exchange prototype builds on the cloud provisioning infrastructure of the ERDDAP prototype as described in the previous section. It makes use of an AMQP based message-passing middleware that provides reliable communication between system components The Data Exchange includes a Ferret THREDDS server as part of a distributed architecture deployed on the Amazon EC2 cloud, as depicted in Fig. 7. The web frontend and catalog components are separated from the server-side dataset caching and indexing components as well as from server-side processing functions. Users can apply Ferret functions to manipulate cloud-cached model data structured in form of rectilinear and curvilinear grids, for instance to subset and time average. A wide range of operations and grid types (including unstructured) is supported through server-side function via the GridFields library [14].

Users can access the prototype using their Matlab environments and via the web browser. Matlab integration with the Data Exchange prototype is provided via the Matlab Structs Tool [15]. This provides
access to any DAP-accessible dataset, which includes Ferret results. A small amount of Matlab code provides a simple interface for creating server-side Ferret programs from within Matlab scripts.

Based on the F-TDS example, we plan to create a similar TDS/GridFields plugin and corresponding Matlab code, so that scientists can more easily work with remote datasets including unstructured grids.

We are also investigating how to add Data Exchange support to existing systems. SOCKS or HTTP proxies, modified DAP libraries are both possible and being considered. Proxies would allow the flexible deployment of cloud-based and local caches without changing working programs.

Fig. 6 shows an overview of the logical architecture of the Data Exchange with its constituent components.

![Fig. 6. Data Exchange prototype logical architecture](image)

### 3.4 Data Exchange Implementation

Fig. 7 shows a depiction of the current distributed component design of the Data Exchange prototype implementation, in the context of a use scenario.
The main components of the Data Exchange implementation are:

- Proxy is a lightweight HTTP proxy that takes the users’ DAP requests and forwards them to the controller.
- The fetcher is the mirror of the proxy, and pulls DAP data and sends it to the proxy and/or persister as directed.
- The persister and cache save and serve data, respectively.
- The controller handles coordination between all the components.
- The attribute store maintains the list of datasets and their status. (Actually, it does more than that, but from the DX point of view that’s its role.)
- The management interface is a web-based app for viewing, registering and deleting datasets, and for setting caching policy.

In the following, we illustrate the Data Exchange prototype function by providing message sequences that show how the components communicate, and the time sequence of operations. This is complex, but simpler to understand than text describing the same operations. These diagrams are the most concise and precise way to describe the function of the components of the data exchange. They do not represent the architecture of the data management subsystem. The Data Exchange design is still evolving in collaboration with the architecture team.

Fig. 8 shows the sequence of messages on a cache miss. User asks for a random chunk of DAP data, which is then used as a trigger to download the entire dataset.
Fig. 8. Data Exchange prototype cache miss sequence

Fig. 9 shows the message sequence of a cache hit. This shows a cache hit. Parts of this are still sketchy and need to be investigated further.

Fig. 9. Data Exchange prototype cache hit sequence

3.5 Results and lessons learned

The separation of storage (caching) and processing components within the architecture enables automatic dynamic provisioning of cloud resources based on demand. When users register new datasets that need to be cached within the cloud infrastructure, new storage resources are elastically provisioned within sec-
onds contingent on policy. Similarly, as new requests for server-side processing arrive from users, potentially requiring significant amount of CPU cycles, new virtual servers are provisioned within seconds and removed once processing completes.

The ability to split the ERDDAP application into several independently-scalable pieces, and the relative inability of some other software packages to be similarly decomposed, clearly indicates the importance of software factorization during development, and of considering what factorization choices are best to enable independent modules to either scale independently or be coupled in a cloud-computing environment.

The results from the ERDDAP and Data Exchange prototypes are very encouraging and valuable. Through the application of cloud provisioning technologies it is possible to deploy larger-scale distributed applications with little effort within a short time; such applications show a high reliability and resilience to failure and they can adapt to load and demand elastically. OOI is currently implementing a further advanced prototype for elastic scaling of distributed applications on demand using resources from multiple clouds, leading to a comprehensive cloud execution infrastructure, the technological basis for the future OOI CI Common Execution Infrastructure subsystem.

Providing a robust data distribution infrastructure is of substantial value for the ocean modeling communities. The availability of an easy to use, reliable and flexible community infrastructure that goes beyond current capabilities provides immediate benefit to groups with fewer available resources; it also provide infrastructure operators and sponsors with the possibility to apply economies of scale to a central infrastructure component, leading to lower cost of operation and optimized resource utilization. In the mid-term, such an infrastructure also increases the drive towards stronger interoperability of data distribution technologies and data representation formats.

For the OOI, an early adoption of a community infrastructure based on the prototyped technologies is of high value, because it promises to lead to earlier and larger community acceptance of future OOI CI infrastructure elements and technologies. In the short-term, the experiences gained with the described prototypes are an effective means for technology risk mitigation prior to the OOI construction period.

IOOS is particularly interested in elements of the NSF OOI CI research that can be used in an operational context as early as possible. The success of the cloud deployment prototyping suggests that popular ocean data providers should transmit a copy of their data to the cloud for general access and distribution, while retaining the master copy locally for limited access. The authors anticipate expanding the OOI CI and IOOS collaboration via a series of demonstration activities as their mutual activities mature. The benefits of such a collaboration based on the described technologies promise to extend well beyond the scope of the DIF.

4 Attribute Store

The Attribute Store is a generic repository of information organized around key + value pairs. It has developed under the umbrella of the Data Exchange prototype but is an independent component of its own. It is the basis of future DM subsystem data and information repositories, as for instance applied in form of the distributed state management service of the COI subsystem and the Dataset registry of the Data Management subsystem.

The semantics of the keys in the Attribute Store (including any form of addressing), and values (including any form of state related information), and hashing of keys to support load balancing (e.g., place half of the keys on a server, and the other half on another) is outside the scope of the attribute store. Main purpose: fast, reliable data storage and retrieval for lightweight data elements (not intended to offer the flexibility of a full-blown SQL engine). Examples include identity management, dataset metadata, etc.
The Attribute Store has three main constituents: (a) Repository to store the actual information; (b) Command Processor to receive, interpret, and then execute commands from its environment onto the information stored into the Repository; and (c) Specification, which describes the capabilities of the Repository and how to match stored entities. The Command Processor operates with a Command Set composed of a set of Commands, including Read, Write, Update, Query and optionally Search (content based). The way of executing the commands depends on the Specification and the capabilities provided by the underlying Repository. At minimum the Lookup Specification describes the way to match entities in the Repository such as string based match (Atom) or a regular expression (Composite of "special" Atoms such as wildcards, patterns, etc).

The core interaction pattern between the Attribute Store and an Application (any kind) is via a lightweight request / response pattern, where the request contains a Command with optional arguments, whereas the response contains the outcome of executing that Command. (see Fig. 11)
The Semantic Prototype for the OOI Cyberinfrastructure, developed on September to December 2009, demonstrated a number of end-to-end capabilities that can semantically enable the OOI. The prototype advanced technologies to expose the meaning of, and enable relations between, concepts used by OOI system components.

The semantic prototype examined an existing collection of data sets, which are already well structured and metadata-enabled. The data sets were available in NetCDF format with standard metadata, however the metadata elements and content varied across communities. Fig. 13 exemplifies the need to apply semantic mediation strategies in OOI CI.

Within the prototype, semantic content that can be exploited and improved was identified for better use of these data sets in OOI applications. Also, the following end solutions were investigated: search capabilit-
ties, automatic semantic indexing of data sets, workflows for modifying existing reference vocabularies, evaluation and tagging of data sets, and validation of existing content for conformity with specifications.

Various technologies were investigated including Semantic wikis, the MMI ontology registry and repository, faceted browser, and metadata catalogs. Ontologies, such as Virtual Solar Terrestrial Observatory (VSTO) and OGC Observation and Measurements (O&M), were investigated for the purpose of being used as core vocabularies.

5.1 Introduction to the Semantic Framework

5.1.1 Semantic heterogeneity and categorizations—scenario

There are several places within OOI CI where semantic services can be used. For this prototype the scenario, where a scientist subscribes to data was advanced. If the CI system is going to provide discovery and notification services for data of interest to users, the system will need to represent the metadata content in homogenous ways. For this to happen the OOI CI system needs to perform metadata mappings and controlled vocabulary mappings, and use them accordingly. The prototype focused on advancing and better understanding of the following:

- Managing of controlled vocabularies and their mappings.
- A system that provides processing of metadata and a faceted search-like interface that will allow users to perform a search and subscribe to that search.

The scenario is well described in the OOI CI Concept of Operations (OOI Document 2115-00002, Version 0r08 2006.05.11", Section B: Getting and Using Products, page 6).

Fortunately, the ORION systems all support a straightforward "publish and subscribe" model for distributing data. A data user can subscribe to a data stream, asking to receive any new data as soon as they is measured by an instrument. In this case, POIM has subscribed to all of the variables it needs and executes whenever a new value is received for any of them. If data arrive while POIM is running, it will cache them, finish its current execution, and start a new cycle with the cached data. Naturally, the data format for some of the sensors is not what POIM requires. Data transformation can be accomplished by the ORION cyberinfrastructure services. For example, when setting up the original subscription request, Dr. Chu asked for the data to be sent in the units that his program requires. The infrastructure services understand how to translate between instrument units and user units, and do so automatically. The default output format for the data (XML-encoded ASCII data values for this type of request) is provided by each observatory using common ORION cyberinfrastructure software.

Although Dr. Chu didn't realize it, when he originally asked for 'oxygen' data, similar mediation services took care of translating the original language of the observatory instruments, which in one case called the data "O2" and in another case "oxygn", to the more general term oxygen. These mappings rely on semantic ontologies, tools, and services established by other organizations, and they specify how any instrumented variable on ORION corresponds to the more standard vocabularies like GCMD and COARDS/CF.

5.1.2 Risk Reduction

The systemic risk posed by the need for semantic interoperability is categorized into two risk factors. Their description, consequences, and tasks to mitigate them are further explained in the Risk Matrix maintained by the OOI CI project. The following risk descriptions highlight the goals for this prototype.
Comprehensive Semantic Framework

**Risk description:** If a comprehensive semantic framework cannot be devised then data interoperability between communities using different vocabularies will be limited. The ability of a broad community of ocean scientists to utilize many types of data requires the ability to mediate diverse data types. A powerful way to provide this is to allow actors utilizing different vocabularies to have a common semantic understanding.

**Starting status**
- Likelihood = 3 = Possible
- Consequence = 4 = (Technical) Marginally acceptable, barely able to perform needed science. Impact of 5-10%
- Rating = medium - some disruption is likely to occur.

**Ending Status Goal:**
- Likelihood = 1 = Remote
- Consequence = 3 = (Technical) Acceptable, with significant reduction capability. Impact of 1-5%
- Rating = low - minimum impact

**Task:** Realize a prototype framework by designing / selecting and testing selected components as follows:
- Semantic Registry
- Semantic Enabled User interface - faceted search
- Metadata enhancer - semantic wiki
- Data2knowledge converter
- Data / Metadata Harvester
- Service Registry

Shared Domain Vocabularies

**Risk description:** If a core set of domain vocabularies are not identified, then the cost of mapping between vocabularies grows exponentially. Domain vocabularies allow domain members utilize common semantics. The OOI will include multiple domains, so vocabularies for the superset will need to be identified to facilitate mapping across all domains.

**Starting status:**
- Likelihood = 3 = Possible
- Consequence = 5 = (Technical) Unacceptable, cannot achieve key team or major program milestone. Impact >10%
- Rating = high - major disruption is likely to occur.

**Ending Status Goal (Task C was no detailed in the Risk register. It is assumed that it refers to ontology models prototyping):**
- Likelihood = 3 = Possible
- Consequence = 4 = (Technical) Marginally acceptable, barely able to perform needed science. Impact of 5-10%
- Rating = medium - some disruption is likely to occur.

**Task:** Develop / adopt superset ontology, with the following initial candidates:
- OGC Observations and Measurement (OM)
- WaterML
- SWEET
- VSTO
5.1.3 Summary Semantic Framework

A set of components was identified to be prototyped in the semantic framework. This included a harvester, a semantic wiki, a faceted browser, an ontology registry (MMI OOR), and a metadata registry (ebRIM), see Fig. 14. The current and desire relation between these components are described in the next two UML diagrams. Detailed explanations of these components are provided in the next section.

Current Components:

Fig. 14. Semantic Framework Integration Prototype current components
Desired Components:

![Diagram of desired components]

5.2 Components

5.2.1 MMI registry

The MMI repository provides the ability to register ontologies, discover them by browsing, retrieve them, and verifying consistency with original contents. The functionality has been enabled for programmatic access to the registry, including demonstration code that other client applications can easily use (directly or as a template). Sequence diagrams and screencasts were created to describe the main operations related with these capabilities.

The intended functionality of the prototype was as follows:

- Allow ontologies to be registered and unregistered against a given graph. Allow rules to be registered and unregistered to the targeted graph 'OOISP'
- Allow to query the 'OOISP' graph via SPARQL / HTTP GET and POST
- Provide utilities to export simple vocabularies to ontologies
- Provide utilities to edit/create relations among resources in the graph, in particular to the selected domain ontology
- Provide the hook for other tools to perform semantic mediation via a centralized system (MMI ORR)

The essential functionality described above is provided, except for the target graph feature, that is, the main MMI ORR graph in the back-end is always the graph affected by the operations.
5.2.2 Harvester

Harvester processes metadata for different data sources and stores it in one homogenous model. Having the data in one model will allow other components (e.g. faceted browser) to query the metadata based on that model. Note that the Harvester was important for this demonstration, but is not central to the Semantic Framework itself.

A functioning prototype harvester has been created that generates either RDF formatted metadata or ncML-G formatted metadata. The metadata is then translated by the harvester into the ISO 19139 metadata model which is required by Geonetwork open source – ebXML and Catalog Services for the Web (CSW). In summary, three types of harvester functions were advanced:

1. Harvest metadata and convert to RDF -> triple store -> Faceted Browser
2. Harvest metadata to Geonetwork (access via CSW) -> Convert to RDF (access by CSW) -> Faceted Browser
3. Harvest metadata using the work done by BlueNet MEST-> Convert to RDF (access by CSW) -
   > Faceted Browser

5.2.3 Faceted Browser

The faceted browser is a user interface that provides clean categorization (facets) for data sources. The faceted browser presents a semantically coherent view of the data sources, where the semantic heterogeneities, such as different representations for one concept, have been resolved by an underline inference engine.

The functionality advance was as follows (Fig. 16):

- Users were provided with a faceted browse interface where the user can search for datasets by making selections on point-of-contact organizations and parameters related to the available dataset.
- An information panel will show properties of the selected dataset/Observation - this includes name, time coverage, abstract, point-of-contact name, email, and organization, and observed parameters.

A basic browser (similar to that created for VSTO) was implemented, showing some of the inferences and relationships that could be incorporated into this kind of user interface once a semantic framework was realized. As a result of the VSTO project, other faceted browser projects, and this prototype, we recommend that OOI CI (a) incorporate into its user interface designs the concept of faceted browsing, (b) anticipate the need for semantic infrastructure and technologies in OOI to support such user-facing tools, and (c) allow for more detailed study in later releases of the particular semantic concepts that are fundamental for accessing OOI data products and other products.
5.2.4 Metadata Registry

The metadata registry will be responsible for storing metadata from various sources, allowing other components to retrieve metadata in a uniform way (e.g. CSW interface / ISO 19115/19139). This component is not central to the semantic risks, but it was needed for the prototype, and the evaluations will contribute to the technology evaluations for data management and other repositories.

The following technologies were investigated: Geonetwork Open Source, GI-CAT, Buddata ERGO ebRIM implementation, and an earlier splinter of Geonetwork that implemented a THREDDS harvester and 19139 converter. None of the available implementations was quite sufficient to meet the needs. Geonetwork 2.x does not have ebRIM support (expected in 3.0) nor a THREDDS harvester; GI-CAT does not persist harvested data; and Buddata is at early development stage.

As all the above technologies are being further developed (with new releases published during the prototype work), they should be monitored further. It is expected convergence of the key features of the repositories, on persistent database backend, support for more harvesters, and ebRIM. As a possible model, more than one repository can be setup, integrating metadata harvested by another repository via CSW (e.g. the Geonetwork - GI-CAT bridge). For the prototype work, Geonetwork open source was preferred due to its persistence within a database (postgress), the activity community behind it and its development friendly access interface implementation.

5.2.5 Community Collaborative Ontology Editor (example: Semantic Media Wiki)

The objective of the community collaborative ontology editor in the prototype is to provide an intuitive interface for scientists to browse ontologies, make corrections while maintaining versioning and provenance, and to export selected ontologies. The editor instance may also export ontologies to another software component, such as a metadata repository or an ontology registry. This capability will be essential to encourage the required social collaborations on vocabularies that will be required for satisfactory community discovery and use of the OOI CI data products.
While the general requirement was for a community ontology editor, the Semantic MediaWiki solution received the most attention during the prototype, as it was deemed a particularly mature technology. Due to results obtained with this product, the evaluation was expanded to other technologies as well (see below).

The Semantic MediaWiki was instantiated in the Amazon cloud at: http://ec2-75-101-200-153.compute-1.amazonaws.com/mediawiki/index.php/Main_Page. And an additional navigation interface has been added at http://water.sdsc.edu:7788/demo/CUAHSI/index.html (see Figure 1), which presents a rich set of hydrologic ontologies. It was use python and the rdflib library to parse the ontologies into a RDF triples. The python wikipedia robot framework was use to generate, delete, and edit pages on the semantic mediawiki instance.

The startree application was configured with a file that specifies the nodes and edges of the tree. The application won't work if the nodes do not form a fully connected tree; therefore, a python script was written to generate the nodes and edges from the ontology. The script ensured that the tree is fully connected.

![Fig. 17. A snapshot of Semantic MediaWiki with Startree Navigation](http://www.oceanobservatories.org/spaces/download/attachments/20087142/SMW_startree.png)

Using the Semantic MediaWiki as an ontology editor presents the following challenges:

1. The process of exporting the edited ontology from Media Wiki to the original OWL is not straightforward. Doing so will require reconciliation of the general RDF model used by Media Wiki with the original OWL file. The modified ontology can then be reloaded with the MMI ontology registry and repository.

2. When properly configured, an end user can edit the ontology by navigating to the entities wiki page and clicking an edit with form. The form provides text boxes for the end user to fill in missing information about the ontology. However, enabling this functionality is not straightforward and there is a need to create templates for each type of resource in the ontology.
5.3 Technology Evaluation – Core Ontology

Metadata models, metadata instances, rules, facets (user categories), and domain subjects can all be represented in different languages, for example UML Class models, XML schemas, configuration files etc. If all of these were represented in one language it will be easier to configure and access the mapping between metadata models, rules for facets generation, and relations with controlled vocabularies. The common language representation used in this prototype was RDF and OWL.

The Ontologies are available in the OOI GIT repository git@amoeba.ucsd.edu:cissemanticprototype.git. Description of the ontologies can be found in the readme.txt file in that branch, which contains the following:

- cdm.owl - ontology representing the content data model (CF Metadata Conventions)
- om.owl - ontology representing the Observation and Measurement (O&M) model
- cf-parameters.owl - ontology representing CF parameters
- fui.owl - contains concepts for categories in the faceted browser
- fui.rules - rules to enable links with CDM, OM and FUI
- mapping-cf-ooi-parameters.rdf - mapping of CF and OOI terms

The selected core ontology was O&M, which is the most comprehensive standard model of observations. Other projects such as NSF SONET [24] are also adopting it. Note that this ontology does not include representation of domain subjects (e.g., parameters), which is perhaps the most challenging semantic concern for OOI (because it will require social engagement to address).

The main metadata fields from the CF metadata conventions [25] that are suitable to use in the semantic framework prototype are:
- title
- institution
- source
- history
- comment
- variable

5.3.1 Technology Evaluation - Ontology Editors

The semantic framework requires easy to use tools to manage ontologies. These include ontology editors. Although some editing capability is already available at the MMI OOR, it was important to also evaluate the ability to make changes to existing vocabularies with other tools that are more likely to obtain social adoption in a collaborative context.

The technologies to be evaluated included SemanticMediaWiki, Collaborative Protégé and Knoodle. The criteria considered were ease of modification, ability to collaborate, fidelity of transactions (are all original data in vocabulary identical after storage and retrieval?), level of metadata kept for each transaction, and interoperability with an ontology repository.

5.3.2 Collaborative Protege

The Protege-OWL editor (http://protege.stanford.edu/overview/protege-owl.html) is a Protege extension that supports OWL. From the web site:

The Protégé-OWL editor enables users to:
- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties, and SWRL rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoners such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup
WebProtege (http://protegewiki.stanford.edu/index.php/WebProtege) is a lightweight web-based ontology editor. It is a client of "Collaborative Protege" server, a Protege extension for collaborative ontology editing and annotation. WebProtege 0.5 Alpha is dated 8/14/2009

5.3.3 With respect to our review criteria:

- Ease of modification: very good, though requires understanding of ontology concepts
- Ability to collaborate: the collaborative version (and the web client) supports projects, logins, annotations, voting, notification and version comparison.
- Fidelity of transactions: transactions are accessible per triple.
- Metadata kept for each transaction: yes, with undo capability;
- Interoperability with repository: yes, can read from URLs both OWL and RDF.

5.3.4 Semantic MediaWiki

This semantic wiki allows users to browse, edit, and collaboratively share ontologies (see http://smwforum.ontoprise.com/smwforum/index.php/Main_Page). User accounts are created to login to the wiki. The administrator can restrict a user's ability to only edit certain pages. The content of the wiki is stored in a MySQL database. The database keeps track of the history of revisions as well as guarantees fidelity of transactions. The content of the semantic wiki can be exported to RDF triples. If the original ontology is stored in OWL there may be difficulty in mapping the exported RDF triples back to OWL.

5.3.5 With respect to our review criteria:

- Ease of modification: very easy (Wiki), preferred by some for this reason. Templates add flexibility in customizing for particular uses.
- Ability to collaborate: as a Wiki, open for collaborative editing. Has user management, access restrictions, annotations and notifications, no special mechanism for projects.
- Fidelity of transactions: managed by MySQL
- Metadata kept for each transaction: yes, and keeps history for undo
- Interoperability with repository: can import OWL, but currently difficult to export to the original content imported.

5.3.6 Knoodle

Knodeel (http://www.knodeel.com/) is a web service built for the express purpose of supporting collaborative development of ontologies. Some of the advertised features:

- Cloud-based; free, but not open source.
- Content organized in "communities"
- A community can have a wiki space and multiple vocabularies (ontologies)
- Allow ontology editing, import/export
- HTTP access, including updating of ontologies
- Every vocabulary is a SPARQL endpoint
- Comment thread per ontology, which are accessible via SPARQL
- Exporting - will export ontology and wiki pages
- Template creation - customize forms
- Permissions to keep content private or public to individual members within a community and non members.

Some observations:

- No rules/inference engine in the free version (but "coming soon") - The licensed version MyKnodeel provides rules capability via Krule.
• Intuitive membership management and assignment of permissions
• Limited SPARQL support (even though they provide a wizard to define queries)
• "Terms of use" page (required reading to using knoodle) not available (2009-12-18)
• "Graph" view does not work; upon saving, a blank page is shown sometimes.

5.3.7 With respect to our review criteria:
• Ease of modification: easy, wiki based.
• Ability to collaborate: open for collaborative editing. User management, permissions, annotations. RSS feeds.
• Fidelity of transactions: good
• Metadata kept for each transaction: history for each wiki page is kept.
• Interoperability with repository: Vocabularies can only be uploaded from local file (not from remote URL, which could be used to import an MMI ontology).

5.4 Future Development Strategies
The following strategies, particularly items 2 and 3 below, have long lead times for full achievement. Thus, they should be pursued by OOI CI even during Release 1, although in many cases they will not be fully adopted or leveraged until Release 3 or later.

1. Define the expected use of semantic mediation solutions in the OOI architecture.
   • Data ingest: As data arrives from any external source -- sensor, model, data archive -- it must be characterized in a form OOI can understand. The relevant semantic activity is to take the metadata (metadata elements and vocabulary terms) used in the original data -- describing topic areas like keywords, units, data processing level, data source type, etc. -- and represent those in an OOI-default semantic model and vocabulary. If no such OOI-default semantic vocabulary exists—for example, it is unlikely that a single science parameters vocabulary will be comprehensive enough to serve as an OOI standard—then leverage existing community vocabularies and mappings to characterize the metadata appropriately.
   
   • Data output: As data is provided to any external client, it is likely that the client would like to see the data (and its metadata) in a format compatible with that client. (For example, GCMD wants to see GCMD keywords for its metadata catalog; many models want to see CF standard variable names.) The OOI needs to be able to convert its internal semantic representations to the externally desired semantic language, or model.
   
   • Any OOI CI resource that needs to be properly defined managed and shared.
      - For example, roles management. Let's say OOI CI has a large set of instances of roles, like Education: Teacher, organized into major groups. To support broader use and edition of this information within and outside the system, it will be useful if this information can be kept, and used, as a hierarchy, available as part of the main OOI knowledge base. This allows software to understand that the Teacher is an educational position, and that Education:Teacher:Teacher_K-12 is still a Teacher, but different than Teacher_Undergraduate.
      - For example, process types. Internal components like Process Execution (data reprocessing) or Presentation: The data in the system may need to be converted or transformed to produce a new product, or to present the existing product. Algorithms in that process may evaluate what to do based on semantic information in the original, for example the exact meaning of an error flag may change the way the corresponding data point is used.
To the extent OOI CI will be using keywords or code lists to name and control its operational behaviors, it will be valuable to document and manage those concepts using a controlled semantic framework.

2. Advance community ontology building by integrating state-of-the-art tools while advancing the MMI registry. In the prototype the core functionality of the MMI ORR registry was tested and advanced, and this appears to be an appropriate system for managing OOI CI-related vocabularies. There is a need to provide better tools for the community to engage in creating, publishing, and advancing these vocabularies. Needed capabilities include creating of groups, tracking changes, enabling permissions, enabling resolution mechanism (e.g. voting), and enabling discussions. Community adoption of this capability will be essential to the creation of effective community vocabularies and mappings, that OOI CI can then leverage to present its data products. Thus, investigation of integration of MMI's ORR with other tools such as Knode and Collaborative protégé is recommended.

3. Advance community ontology building by creating a community-supported process for capturing and improving ontologies. As of today, there are many relevant existing community vocabularies, and communities interested in making new vocabularies, that will be needed for OOI CI communities and systems. Yet there is no widely publicized method or training available for encouraging this process. The Marine Metadata project has held several workshops to engage the community in similar processes, developed tools, guides, and templates for holding such workshops, and is in a good position to encourage wide adoption. We recommend that the MMI project be solicited to further this process with regular workshops and other activities, so as to encourage a common, widespread community practice of vocabulary creation and exchange.

6 Hyrax and GridFields Integration Prototype

The University of Washington, OPeNDAP, and The University of California at San Diego (UCSD) are building a pilot demonstration of interoperability between GridFields and Hyrax integrated in the OOI Data Exchange. The GridFields capabilities are demonstrated by a subsetting operation over an unstructured grid data source returning a UGRID-compliant dataset. UGRID is a community standard which is still in development but is the leading candidate for an unstructured data model built on the NetCDF 3.0 file format. The UGRID effort is an outgrowth of the 2006 Community Standards for Unstructured Grids workshop.

To integrate Hyrax with the OOI infrastructure it is necessary to support for the DAP protocol over AMQP. A prototype of Hyrax which supports transfer over AMQP was created by adding a new front-end to the server that can act as an AMQP client, reading information from an AMQP queue. Hyrax has an overall architecture that already supports this. Figure one shows a high-level view of the Hyrax architecture. The BES is the part f Hyrax that builds the bodies of a DAP response. The front-end (the OLFS) contains a set of handlers that respond to requests made using HTTP. Based on the request, the OLFS sends commands over a stateful connection to the BES asking it to make the correct response. Generally, the OLFS will have to parse the request URL and pass information from that URL to the BES. Even though the OLFS is designed to support several different 'protocols' like DAP or THREDDS, it is capable of responding to HTTP only (it is a Java Servlet; see Server Dispatch Operations for information about the OLFS design, implementation and extension capabilities). Thus, it makes the most sense to build a new front-end dedicated to AMQP.

While the architecture chosen to add support for AMQP is very important, other considerations are also critical to the success of the overall effort to run DAP over AMQP. One is the mapping between different DAP versions to AMQP. Since DAP was designed with HTTP in mind, how DAP and AMQP can best be matched merits serious consideration. This will entail looking at the current DAP implementation along with the evolving DAP, version 4, specification and its implementation.
The demo will service an OPeNDAP request over unstructured grids. The request will perform simple subsetting. The returned stream will be compliant with the UGRID model being developed by Brian Blanton, Bill Howe, David Stuebe and Rich Signell. The request URL may include a custom dispatch handler call of the form

```plaintext
...&subset(expr)
```

where `expr` is a conditional expression involving the attributes of the source grid. For example, a bounding box expression "x between 29000 and 31000 and y between 28000 and 31000" can be encoded as follows:

```plaintext
...&subset(x<31000,x>29000,y<31000,28000>y)
```

The Hyrax server translates this expression into an equivalent GridField query, evaluating it, formatting the result, and returning the result using the UGRID format.

### 6.1 Grid Fields as a Backend Service

#### 6.1.1 Why Unstructured Grids are Complicated

Why are unstructured grids so difficult to work with? Consider Fig. 18. At left, the highlighted cells of the structured grid represent a region of interest to a user. These cells may be addressed by a simple range query over world coordinates (latitude and longitude), which is trivially translated to a range query over computational coordinates in the corresponding representation in memory or on disk. We say that the two coordinate systems are spatially coherent — cells that are near each other in world coordinates also tend to be near each other in computational coordinates. At right, a user-selected region from an unstructured grid consists of four triangular cells. These four cells may appear anywhere in the overall representation. In general, representations are not spatially coherent — knowing where a cell is in world coordinates gives no hint as to where to find it in computational coordinates.

This simple fact has profound consequences for interoperability and performance. The algorithms developed to operate on unstructured grids use a variety of tricks and conventions; we say that they are tightly coupled to representation details such as cell order and implicit conventions for expressing neighborhood relationships. For example, particle tracking algorithms must access a local neighborhood of velocity values to determine where a moving particle will go next. Therefore, the algorithm must gather up the velocities nearby to the particle's current position. If these velocities are nearby in the representation, then lookup is easy, and cache performance is good. However, if the velocities could appear anywhere in the representation, then the algorithm must search for them, or build some kind of index beforehand, or do some form of guess-and-check to compute where the particle goes next. In practice, we find all of these solutions and many more, none of which are compatible with each other's representation. None of these solutions have anything to do with the underlying science of the problem. Rather, they are consequences of physical data dependence---an artificial coupling between algorithm and representation. The UGRID software, together with the underlying formalisms of the GridFields model, separates what needs to be done, from how to do it in order to achieve interoperability.
6.1.2 Mapping the DAP data model to GridFields Data model

The following figure (Fig. 19) shows the mapping between the major components of an unstructured dataset in NetCDF, DAP and GridFields. This most important feature is the expression of the array that lists the nodes surrounding each element. In this case it is the variable Grid1. While the content expressed in the header information below is not complete, the data models are isomorphic. The final implementation will be able to read a UGrid NetCDF file from disk into Hyrax which converts it into DAP data objects, create a subset of that dataset using the gridfields data model and then send the results to a client using the DAP encoding.

```groovy
Dataset {
  Grid {
    Face02 gridFieldsConnect="true";
    nCell(=1570);
    nConnect(=44);
  }
  Variable "node_connect" {  
    gridFieldsConnect="true";
    nCell(=1570);
    nConnect(=44);
  }
  Variable "node" {  
    gridFieldsConnect="true";
    nCell(=1570);
    nConnect(=44);
  }
}
```

Fig. 19. Mapping between major components in GridFields prototype

6.2 Design of the Hyrax - Gridfields Service

Gridfields is implemented as a back end for unstructured subsetting in the Hyrax data server using a custom function call. The Hyrax server is designed to handle custom functions to provide an extensible option for the DAP query string. The input argument is a DAP data structure which contains references to the variables and attributes in the data set. The output argument is new DAP data structure that is constructed as the response. Using the Hyrax infrastructure the attributes of the dataset are read into Grid-
Fields. From this information the correct procedure to complete the request, and the components of the dataset that are required, are determined. Those components are loaded from the file and processed. The calling function then constructs a DAP structure which contains the response and the function returns.

6.3 Separating protocol from transport in AMQP

Fig. 20 shows how a AMQP module could be added to work with Hyrax. In the figure, a single BES daemon is shown as being shared by both the OLFS – the component of the Hyrax server that implements the DAP-over-HTTP – and the proposed AMQP front end. In fact, while this is certainly possible, it is not the only way that the proposed AMQP front end could be used with the BES. Other combinations of the AMQP front end and the BES include exclusive use of one or more BES daemons by the AMQP front end or both the AMQP and OLFS components accessing a set of BES daemons spread across several hosts.

To leverage existing tools such as SOAP over AMQP, it makes sense to look at how OPeNDAP has implemented the SOAP bindings for DAP. The SOAP interface to Hyrax differs from the DAP-over-pure-HTTP interface in that the latter uses HTTP’s GET method and encodes the complete request in a URL. This is important for clients because handling the request like this makes for a very transparent request ‘object’ that virtually every client can make, edit and store. Also, because HTTP is so ubiquitous, this form of a request can be ‘dereferenced’ (i.e., the data can be accessed) by many clients (e.g., Excel) without modification. However, the SOAP interface presents some important advantages. The most useful in this context is that SOAP is a formalism that can be applied to many different transport protocols. When OPeNDAP designed the DAP-over-SOAP implementation, we made sure to take advantage of the ability to bundle many discrete DAP requests into one SOAP envelope. This makes for a powerful way to economize on network interactions, if, for example, a client knows that it will need to make a series of requests of one server.

One issue, however, that is likely to play a role in DAP over AMQP that doesn't show up in the current SOAP interface is that DAP4 is now much farther along than when the SOAP interface software was written. The feature of DAP4 most important to this project is that DAP4 no longer relies on HTTP headers as the sole way to return certain information. Instead, all information about a response is contained in the body of the response and some information is also contained in HTTP response headers to simplify writing HTTP clients and/or working with DAP2-only clients. So, for example, the information about the version of DAP used to build a particular response is now part of the response body (in the <Dataset> element) and in the HTTP response header XDAP. This means that HTTP clients can figure out the version before the response document is parsed and other protocols (e.g., AMQP) can get it from the response itself.
6.3.1 How the OLFS connects to the BES

On start-up the OLFS makes a connection to the BES. When the OLFS is started, the BES Daemon (bes-daemon) has already started and bound a well-known port (10002 by default; set in both the BES and OLFS configuration files). The OLFS starts when Tomcat starts or restarts the servlet and initially makes a pool of connections that are TCP socket connections to specific instances of the BES listener (beslistener). When the OLFS gets a request it needs to process using the BES, it checks this pool of connections and picks the next available one. If no connections are available, then a new connection is made unless the maximum number of allowed connections have already been made. In the latter case the request for the next available connection blocks until there is an available connection. The maximum number of connections to the BES, which is really the maximum number of BES listeners (i.e., processes) to make, is set in the OLFS configuration file.

Important points:

1. The current OOI high-level architecture does not allow for direct addressing of machines within the OOI cloud. This will preclude having a front end (regardless or whether it is the OLFS or proposed AMQP front end) use TCP to establish communications with the BES.
2. It will be possible to use the Hyrax architecture within the confines of a single physical machine within the cloud, but this may be a serious limitation given the OOI project’s ambitious goals.

However, there are two possible ways in which the limitation regarding TCP and direct addressing can be overcome. First, TCP can be tunneled over AMQP. Even though this is a course solution at best, it can be used as an interim solution to evaluate the impact of running multiple BES processes on multiple machines in the OOI cloud. The second solution is to modify the front- and back-end components (the AMQP front end and the BES) so that they use AMQP for their interactions as well. While this might seem very complex, it is in fact not that hard. The TCP-based communication between the front and back-end components of Hyrax is already tuned to a messaging-type protocol with each request posed in an
atomic request ‘document.’ Somewhat more complex is the handling of the response from the back-end because it can return a stream of data.

6.3.2 Abstracting the OLFS/BES connection logic
To see how to abstract the connection logic so that the OLFS’s connection pooling and configuration logic can be reused with a different transport protocol, look at the interface for this class. This class implements a simple interface with the methods:

- **Init**: Make the object that holds state for the request
- **Open**: Connect to a new BES listener
- **Send request**: Given that a connection to a server exists, send a request
- **Process response**: Wait for, and then process, a response to a request
- **Close**: Deallocate resources associated with this connection

It will be straightforward to extrapolate this interface to one which either provides for tunneling TCP over AMQP or which uses AMQP directly. In addition, it may also be that RabbitMQ provides a tight enough integration with Java's IPC classes that a more straightforward implantation is possible.

6.3.3 Abstracting the request-response logic of the OLFS

![Diagram of request-response logic](image)

Fig. 21. The current OLFS implementation is tightly coupled with the Servlet classes, especially the HttpServletRequest and Response classes
Fig. 21 shows that the current OLFS implementation is tightly coupled with the Servlet classes, especially the HttpServletRequest and Response classes while Fig. 22 shows how the operation of that software can be modified so that the bulk of the code can be (re)used by both the OLFS and the AMQP front end. The importance of maximizing the code sharing between various front-end components lies in reducing the risk associated with OOI having to maintain software that should rightly be outside of its control. By making these modifications to the OLFS’s software, most of it can be reused by the AMQP front-end and thus, as changes are made to the OLFS/BES interactions, those changes will be inherited by the AMQP front-end too.

Fig. 22. The OLFS implemented using HyraxRequest and HyraxResponse objects. Using these factors out the coupling between Servlet and the OLFS at all but the highest levels of implementation
Fig. 23. The AMQP front end using HyraxRequest and HyraxResponse objects. This uses the same code as the OLFS with the HyraxRequest/Response objects, limiting issues with long-term maintenance

6.3.4 How complex would this be to implement?

Most of the work in defining the set of values to be extracted from the HttpServletRequest object has already been done and resides in the OLFS’s ReqInfo class. This class has eleven (11) methods that return information needed by one or more of the dispatch handlers. Nine (9) of the methods return string values and two return boolean values. Searching within the OLFS source code, there are 128 uses of the HttpServletRequest class (excluding the experiment code and all the classes that provide support for WCS). This corresponds to 23 classes in three packages.

Replacing the HttpServletResponse object may be more complicated, depending on the capabilities of the AMQP Client software base. The servlet response provides methods to write certain specific headers that will preface the response body in HTTP’s pseudo-MIME response document. The response also contains a stream used to write the body of the response, and it allows the OLFS to provide HTTP status responses. It may be that the best approach is to modify the dispatch software so that it builds two things: the response preface material and the response body. Some implementations of the HyraxResponse class would ignore the preface material (e.g., the ContentType header), others might format that in different ways. To determine more about this class’ construction, we need to know what the AMQP client object expects and the facilities it provides.

Additionally, components of the OLFS need access to resources held on the local drive, both within the web applications distribution directory and in the persistent content directory used by the OLFS to hold configuration, logs, and as a place for components to store state information. The OLFS’s ServletUtil class currently provides those local resources paths. This information would also need to be included in the data object that we develop to abstract the HttpServletRequest information.

7 Summary

The three Data Management subsystem oriented prototypes provided substantial insight and refinement for the OOI CI Data Management subsystem architecture. The Data Management prototypes thereby con-
stituted valuable risk mitigation activities leading to increased readiness for OOI CI construction. Substantial technical risks associated with the advanced, transformative OOI CI architecture have been addressed and partially mitigated.

The Data Exchange prototype has demonstrated how to deploy existing data distribution and transformation applications in the cloud as a distributed system based on a message-passing infrastructure. Vertical and horizontal scalability could be demonstrated as a benefit of this architecture. This prototype is also central to seed community adoption of a operational prototype and instrumental in the collaboration with IOOS. It will continue to evolve as part of this collaboration and be succeeded by release 1 of the OOI CI on a much broader scope.

The Attribute Store is a significant dependency of the Data Exchange architecture, design and implementation. Beyond its functional value as a generic key/value store for the Data Exchange, it is prototypical for the description of all OOI CI services, and serves as core component providing persistence and repository services to other prototypes.

The Semantic Framework Integration Prototype illuminated a set of advanced Data Management applications and existing technologies. The prototype integrated state of the art technologies to meet the needs of the science community. Further effort evaluated the semantic requirements of the OOI CI system which lead to a refined architecture for the Data Management system. Most fundamental Data Management services in OOI CI release 1 are informed by semantic mediation, which will be integrated into release 2.

The Hyrax/GridFields Integration Prototype provided insight into high performance science data distribution and processing. It also brought applications and transformations for generic unstructured data-sets into the experience of the OOI CI developers.

The risks targeted by the described prototypes in the Data Management subsystem are:

- **#2226 Comprehensive Semantic Framework.** The Semantic Framework Integration Prototype realizes such a semantic framework by designing, selecting and testing selected components, such as Semantic Registry, Semantic Enabled User interface with faceted search, Metadata enhancer / semantic wiki and Service Registry. This prototype demonstrates the feasibility of integrating semantic mediation technologies in an observatory setting and prepares the OOI CI for integration of these technologies in release 2, by providing requirements and extension interfaces to the Data Management services in release 1.

- **#2227 Shared Domain Vocabularies.** The Semantic Framework Integration Prototype provided a platform for developing or adopting superset ontology, with the initial candidates, such as OGC Observations and Measurement (OM), WaterML, SWEET, VSTO ontologies. The experiences gained in this prototype provide important technical requirements for the implementation of Data Management services and data/metadata model specifications in release 1.

- **#2204 Multiple Technology Integration.** The Data Exchange and the Hyrax/Gridfields prototypes as well as the AttributeStore implementation demonstrate the feasibility of the message-based communication infrastructure as integration fabric for heterogeneous technologies into a consistent system of systems. The integration strategy is based on the availability of a scalable, secure, reliable message-broker infrastructure, and the explicit definition of message-based interactions between services, which encapsulate tools and technologies.

- **#2230 IOOS Interoperability.** The Data Exchange prototype is a joint collaboration between OOI and IOOS in order to achieve interoperability between the future observatories. Alignment on common standards (such as NetCDF, DAP, CF conventions), and tools (THREDDS server) and successful deployment within IOOS user communities reduces the interoperability risk substantially.

- **#2239 User Involvement.** The Data Exchange prototype targets data analysis and numerical modeling communities. It provides a cloud-based data distribution and processing infrastructure with added value capabilities, such as server-side processing of unstructured gridded data via the GridFields operators. Selected target deployment communities out of the IOOS regions have been
identified for the early deployments. The convenience of such deployments and the added value for the community with reduce the risk of user involvement substantially.

In addition to mediating risks directly through demonstrable achievements, the prototypes identified areas that can benefit from further risk reduction activities. One example is the pursuit of target communities for early deployments (addressing User Involvement risk), and the IOOS collaboration represents another ongoing risk reduction activity to address IOOS Interoperability risk. In section 5.4, the semantic prototype activity produced several specific recommendations to further reduce the two semantic risk items. In section 6.3 the OpenDAP team presented specific plans for redesigning the Hyrax front end to better integrate with the OOI architecture based on the lessons from the prototype. By advancing our understanding of needed developments, these prototypes have further improved OOI CI's ability to satisfactorily meet the needs of its user community.

8 References

[16] NOAA ERDDAP. Website http://coastwatch.nfeg.noaa.gov/erddap/
[23] Unidata THREDDS Data Server (TDS). Website http://www.unidata.ucar.edu/projects/THREDDS/